Lab 2

EECS 4312

Jinho Hwang (215240559)

October 30, 2020

Contents

1 Part I

Python scikit-learn package was used to do this lab. To replicate the result, please run **python3 run.py** on each part.

1.1 Result

Android	precision	recall	f1_score
BernoulliNB	0	0	0
ComplementNB	0.111111	0.00411523	0.00793651
GaussianNB	0	0	0
MultinomialNB	0.111111	0.00411523	0.00793651

Figure 1: Part I Android result. Four different Naive Bayes classifiers performance chart with their precision, recall and f1 score on class 1.

Openstack	precision	recall	f1_score
BernoulliNB	0	0	0
ComplementNB	0.210526	0.507463	0.297593
GaussianNB	0.142857	0.00746269	0.0141844
MultinomialNB	0.209375	0.5	0.295154

Figure 2: Part I Openstack result. Four different Naive Bayes classifiers performance chart with their precision, recall and f1 score on class 1.

1.2 Conclusion

Naive Beyes classifier is a simple classifier and it is expected to yield a bad result.

2 Part II

The product of the sets of each possible parameter below were fed into the classifier to create tables. The table contains the parameter name column as well as the f1 score to evaluate each classifier's performance on the changing parameter setting. For example the following table entry:

max_depth	max_leaf_nodes	min_samples_split	f1_score
		32	0.4

Figure 3: Decision tree parameter setting and its performance on Android data set.

is the result of decision tree predicting with parameter **max_depth** being **None** (which means no limit to max depth), **max_leaf_nodes** being **None**, and **min_samples_split** being 32, with the performance(f1_score) 0.4.

2.1 Decision tree classifier

2.1.1 Parameters

```
"max_depth": [None, 8, 4, 2, 1],
"max_leaf_nodes": [None, 2, 4, 8, 16],
"min_samples_split": [2, 4, 8, 16, 32, 64],
```

2.1.2 Table

1. See `./part_II/result/Android_decision_tree.csv` for the Android table result.
2. See `./part_II/result/Openstack_decision_tree.csv` for the Openstack table result.

2.1.3 Best results

max_depth	max_leaf_nodes	min_samples_split	f1_score
		32	0.4

Figure 4: Best decision tree prediction performance on Android data.

max_depth	max_leaf_nodes	min_samples_split	f1_score
		8	0.32

Figure 5: Best decision tree prediction performance on Openstack data.

2.2 Random forest classifier

2.2.1 Parameters

```
"max_depth": [None, 8, 4, 2, 1],
"max_leaf_nodes": [None, 16, 8, 4, 2],
"bootstrap": [True, False],
```

2.2.2 Table

1. See `./part_II/result/Android_random_forest.csv` for the table result.
2. See `./part_II/result/Openstack_random_forest.csv` for the table result.

2.2.3 Best results

max_depth	max_leaf_node	bootstrap	f1_score
		False	0.26

Figure 6: Best random forest prediction performance on Android data.

max_depth	max_leaf_node	bootstrap	f1_score
		False	0.1

Figure 7: Best random forest prediction performance on Openstack data.

2.3 Logistic regression classifier

2.3.1 Parameters

`"max_iter": [1, 10, 100, 1000, 10000, 100000],`

2.3.2 Table

1. See `./part_II/result/Android_logistic_regression.csv` for the table result.
2. See `./part_II/result/Openstack_logistic_regression.csv` for the table result.

2.3.3 Best results

max_iter	f1_score
1	0

Figure 8: Best logistic regression prediction performance on Android data.

max_iter	f1_score
1000	0.03

Figure 9: Best logistic regression prediction performance on Openstack data.

2.4 Conclusion

The best classifier was the **decision tree classifier** with the parameter: no max depth, no max leaf nodes, with 32 minimum sample split.

3 Part III

The new feature added on top of Part II is the bag of word. The bag of word is the 1-gram word vector extracted from the commit message data.

3.1 How was the bag of word added?

The abstract idea of adding the bag of word as the data feature is simple. Import the commit message, clean it, create a bag of word, remove infrequent(≤ 3) word, and merge it to the original feature data.

3.1.1 Importing data

The following function **classify_and_report** takes the classifier with parameters from Part II, takes training file directory, test file directory, and comment file directory, then use the three files to combine them into a feature data set using **add_bag_of_word_feature**. The feature data is fed into the classifier to fit the data and its prediction result is used to create f1 score that is be outputted to ./result.

```
# Part III run.py
def classify_and_report (classifier, train_file, test_file, comment_file):
    # load the comment data as a matrix
    comment_df = pd.read_csv(comment_file, encoding="Windows-1252")

    # load the training data as a matrix
    train_dataset = pd.read_csv(train_file, header=0)

    # load the testing data
    test_dataset = pd.read_csv(test_file, header=0)

    train_dataset, test_dataset = add_bag_of_word_feature(train_dataset, test_dataset, comment_df)

    #...
```

The **add_bag_of_word_feature** takes the train data, test data, and comment data to merge each train and test data with the comment bag of word features by cleaning the comment using **clean_function** and **bag_of_wordify** function.

```
# Part III bagofwords.py
def add_bag_of_word_feature(train_data_df, test_data_df, message_df):
    # Clean messages
    message_df["Comment"] = message_df["Comment"].apply(clean_function)
    # Bag of wordify
    bag_of_word_df = bag_of_wordify(message_df)
    # Insert bag of word entry's respective ID
    bag_of_word_df.insert(0, "Id", train_data_df.loc[:, "Id"])

    # Left merge the bag of word data
    train_data_df = train_data_df.merge(bag_of_word_df, how="left", on="Id")
    test_data_df = test_data_df.merge(bag_of_word_df, how="left", on="Id")

    return train_data_df, test_data_df
```

3.1.2 Cleaning the comment data

The **clean_function** is applied on the commit messages, resulting in "clean" message that does not include any stop words, meaningless words like next line, Unicode, and repetitive ending word.

```
# Part III bagofwords.py
def clean_function(p_string):
    stop_words = set(stopwords.words('english'))
    custom_stop_words = ["", ":", "\\n", "\n", ".", "#", "--", '"', "'", "[", "]", "-", "()", "(", "
    )", '\\"', "*"]

    # These words will be omitted from word tokens
    stop_words = stop_words.union(custom_stop_words)
    # These words will split the token
    splitting_word = ["\\n", "\\u003"]

    # These words and after will be deleted from the committed message.
    repetative_end_word = ["Orig-Change-Id", "Change-Id:", "Signed-off-by"]

    # Words will be lemminized
    lem = WordNetLemmatizer()

```

```
    result = p_string

    for rew in repetative_end_word:
        if rew in result:
            reg_tok = RegexpTokenizer(f"(.*){rew}")
            result = " ".join(reg_tok.tokenize(result))

    for sw in splitting_word:
        result = result.replace(sw, " ")

    word_tokens = word_tokenize(result)

    filtered_sentence = [w for w in word_tokens if not w in stop_words]
    lemminized_setence = [lem.lemmatize(w) for w in filtered_sentence]

    result = " ".join(lemminized_setence)

    return result
```

3.1.3 Creating bag of words and dropping all infrequent word

CountVectorizer from **sklearn** library is used to create the bag of word out of the message data with column [Id, Comment]. The resulting **bag_of_words_df** contains the bag of word count vector with the columns respected to its word.

```
# Part III bagofwords.py
def bag_of_wordify(message_df):
    comments = message_df['Comment']

    count = CountVectorizer()
    bag_of_words_vector = count.fit_transform(comments)

    feature_names = count.get_feature_names()
    bag_of_words_df = pd.DataFrame(bag_of_words_vector.toarray(), columns=feature_names)

    # remove all columns sum <= 3
    bag_of_words_df.drop([col for col, val in bag_of_words_df.sum().iteritems() if val <= 3], axis
=1, inplace=True)
    return bag_of_words_df
```

3.1.4 Merging the bag of word with original feature

The bag of word data with inserted ID is merged with train data and test data (line 11, 12) then these merged data is used to fit the decision tree classifier to give the f1 score result. The sample for such merged data is shown in **Figure 10** below.

```
1  # Part III bagofwords.py
2  def add_bag_of_word_feature(train_data_df, test_data_df, message_df):
3      # Clean messages
4      message_df["Comment"] = message_df["Comment"].apply(clean_function)
5      # Bag of wordify
6      bag_of_word_df = bag_of_wordify(message_df)
7      # Insert bag of word entry's respective ID
8      bag_of_word_df.insert(0, "Id", train_data_df.loc[:, "Id"])
9
10     # Left merge the bag of word data
11     train_data_df = train_data_df.merge(bag_of_word_df, how="left", on="Id")
12     test_data_df = test_data_df.merge(bag_of_word_df, how="left", on="Id")
13
14     return train_data_df, test_data_df
```

3.2 Result

	Id	owner_x	month_of_year	day_of_month	day_of_week	hour_of_day	minute_of_hour	deletion_x	addition_x	changed_file_30	...	zooming	zorder	zoreil	zsh	zte	zygote	%char	%const	%t	%void
0	577	1000392	9	21	3	15	27	36	29	0	...	0	0	0	0	0	0	0	0	0	0
1	579	1000162	9	21	3	19	57	0	5	0	...	0	0	0	0	0	0	0	0	0	0
2	593	1000162	9	21	3	22	2	0	1	0	...	0	0	0	0	0	0	0	0	0	0
3	594	1000162	9	21	3	22	2	0	1	0	...	0	0	0	0	0	0	0	0	0	0
4	595	1000162	9	21	3	22	2	0	8	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6094	22322	1000660	3	12	3	23	47	8	0	0	...	0	0	0	0	0	0	0	0	0	0
6095	22323	1000660	3	12	3	23	47	15	0	0	...	0	0	0	0	0	0	0	0	0	0
6096	22324	1001891	3	13	4	0	37	0	3	1	...	0	0	0	0	0	0	0	0	0	0
6097	22325	1002621	3	13	4	2	41	0	1	1	...	0	0	0	0	0	0	0	0	0	0
6098	22328	1005158	3	13	4	7	59	1	1	1	...	0	0	0	0	0	0	0	0	0	0

6099 rows × 7238 columns

Figure 10: The sample Android training data set with merged bag of committed message and original feature. (Run PartIII/bagofwords.py to get this table)

max_depth	max_leaf_node	min_samples_sp	f1_score
		32	0.25

Figure 11: The Android project prediction with decision tree and bag of word.

max_depth	max_leaf_node	min_samples_sp	f1_score
		32	0.12

Figure 12: The Openstack project prediction with decision tree and bag of word.

3.3 Conclusion

The f1 score after adding the bag of word as a data feature has decreased the f1 score then if it was not added. A lesson here is that adding more data feature for classifier to look at does not necessarily increase the f1 score.